Tutorial- 3

A-1 →

```
While (low <= high)
    mid = low + (high - low) / 2;
    if (arr[mid] == key)
        return true;
    else if (arr[mid] > key)
        high = mid - 1
    else
        low = mid + 1;

return false;
```

A-2 → Iterative insertion Sort

```
for (i to n)
    key = arr[i];
    j = i-1
 while (j <= 0 && key < arr[j])
    arr[j+1] = arr[j];
    j--

 arr[j+1] = key.
```

Recursive Insertion Sort

```
insertion Sort (arr[], n)
    if (n<=1)
            return;
    insertionSort (arr, n-1);
    int last = arr [n-1]
     j = n-2;
     while (j>=0 && arr[j] > last)
     {   arr [j+1] = arr[j];
             j--;
     }
      arr [j+1] = last
```

Insertion Sort is an online sorting Algorithm because it does require the whole array in order to place the element into its correct place all the time.

A-3 →

Bubble Sort	-	O(n^2)
Insertion Sort	-	O(n^2)
Selection Sort	-	O(n^2)
Merge Sort	-	O(nlogn)
Quick Sort	-	O(nlogn) worstcase O(n^2)
Count Sort	-	O(n)

A-4 → Online Sorting → Insertion Sort

Stable Sorting → Merge Sort, Insertion Sort, Bubble Sort, .

Inplace Sorting → Bubble Sort, Insertion sort, Selection Sort.

A-5 → Iterative B.S

```
while (s <= e)
    mid = s+(e-s)/2
    if (arr[mid] == targ.)
            return true;                      O(logn)
    else if (arr[mid] > targ.)
            end = mid-1;
    else
            s = mid+1;
return false;
```

Recursive B.S

```
~~while (s<=e)~~     if (s>e)
                        return false;
    mid = s+(e-s)/2
    if (arr[mid] == targ)
            return true;
    else if (arr[mid] > targ)
            BS(arr, s, mid-1)
    else
            BS(arr, mid+1, e)
```

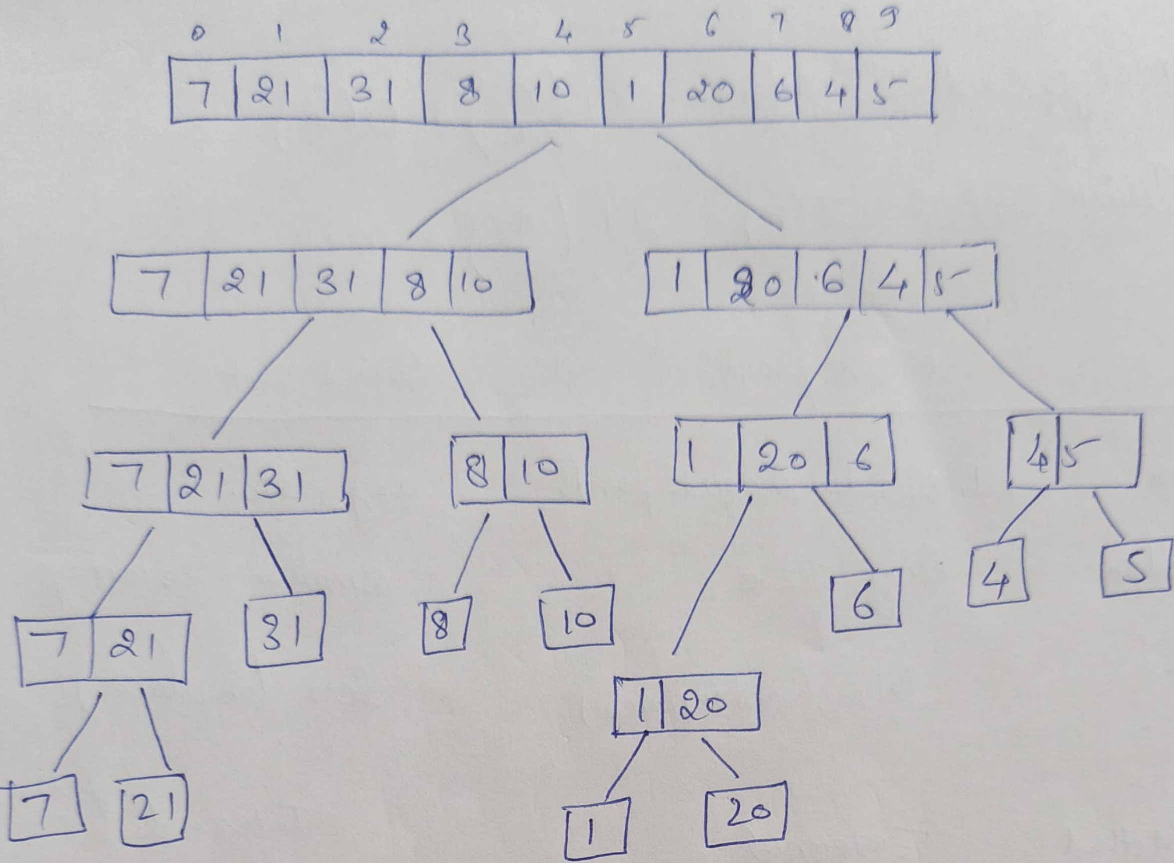
A-6 → $T(n) = T\left(\frac{n}{2}\right) + c$

A-7 →
```
map <int, int> m;
for (i = 0 to arr size )
        if (· tgt - arr[i] ∈ m && equal to ma
             m [arr[i]] = i
        else
             cout << i << " " << m[arr[i]];
```

A-8 → Quicksort is the fastest general purpose sorting algo. So most practical solution. by using Devide and Conquer algorithm.

If stability is important & space is available, mergesort is the best.

A-9 → Inversion Indicates how far or close the array is from being Sorted.



Inversions = 31

A-10 The worst case occeuos when the picked pivot is always on extreme (smallest / largest) element.

This happens when input array is sorted in reverse order. and either first or last element is picked as pivot.

$O(n^2)$

The Best case occeurs when pivot element is middlee or next to the midde element.

$O(n \log n)$

A-11 → Merge Sort : $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Quick Sort : $T(n) = 2T\left(\frac{n}{2}\right) + n+1$

Basis	Merge Sort	Quick Sort
Partition	halved to 2 equal parts	any ratio.
works well on	Any size	Smaller size arrays
additional Space.	More (not inplace)	less (in place)
Sorting Method	External	Internal
Stability	Stable	Not stable

A-12 → Selection Sort can be made stable if instead of swapping, the minimum element in placed in its position without swapping. i.e. by placing no. in its position every element one step forward.

A-13 → We can set a flag one & if after any pass there is no swapping performed. It means the array has been sorted & we can break out of the loop.

A-14 → We will use merge sort because we can devide the 4 GB data into 4 packets of 1GB & sort them separately & combine them later.

Internal Sorting → All the data to be sorted is stored in memory all the time while sorting is in process.

External Sorting → All the data not needed to be on the RAM while sorting.